作业提交服务器：ftp://192.168.134.123

用户名：uploader

密码：sdt7%5252@3

❗注意事项

1. 每题需要提交一个cpp文件，确保cpp文件可以正确编译。cpp文件用题目编号命名。
2. 每题需要提交至少一张运行结果截图。将所有题目的截图放在一个pdf文件中。
3. 将所有cpp文件和一个pdf文件打包成zip文件，用自己的学号命名，并上传至ftp指定文件夹。

   举例说明：比如学号是1001，本次有3个题目，那么最终提交1个1001.zip文件，其中包含3个cpp文件（分别是1.cpp, 2.cpp, 3.cpp）和1个pdf文件，其中pdf文件中包含至少3个运行结果的截图。第1次作业上传至hm1文件夹。

第3次作业

提交时间：2024年5月13日22:00

设计一个员工管理系统，该系统包含如下功能：

1. 添加和解雇员工；
2. 员工晋级和降级；
3. 查看所有员工，包括离职的和在职的员工；
4. 查看所有在职的员工；
5. 查看所有离职的员工。

该系统包含三个部分：Employee类封装单个员工的信息，Database类管理所有员工，单独的UserInterface提供程序的交互接口，其具体实现参见如下的userinterface.cpp。

```cpp
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
#include "employee.hpp"
#include "database.hpp"

//该函数用来初步测试Employee类的实现是否正确，你可以增加更多的代码来进行测试
void test_employee()
{
    using namespace Records;
    Employee e { "Jane", "Doe" };
    e.set_first_name("John");
    e.set_last_name("Doe");
    e.set_employee_number(71);
    e.set_salary(50'000);
    e.promote();
    e.promote(50);
```

```
    e.hire();
    e.display();
}

//该函数用来初步测试Database类的实现是否正确，你可以增加更多的代码来进行测试
void test_database()
{
    using namespace Records;
    Database my_db;

    Employee& e1 { my_db.add_employee("Greg", "Wallis") };
    e1.fire();

    Employee& e2 { my_db.add_employee("Marc", "White") };
    e2.set_salary(100'000);

    Employee& e3 { my_db.add_employee("John", "Doe") };
    e3.set_salary(10'000);
    e3.promote();

    std::cout << "All employees: " << std::endl << std::endl;
    my_db.display_all();

    std::cout << "Current employees: " << std::endl << std::endl;
    my_db.display_current();

    std::cout << "Former employees: " << std::endl << std::endl;
    my_db.display_former();
}

using Records::Database;

//以下四个函数提供了交互接口，并都已经实现
int display_menu();
void do_hire(Database& db);
void do_fire(Database& db);
void do_promote(Database& db);

int main()
{
    Database employee_db;
    bool done { false };
    while (!done) {
        int selection { display_menu() };
        switch (selection) {
            case 0:
                done = true;
                break;
            case 1:
                do_hire(employee_db);
                break;
            case 2:
```

```
                do_fire(employee_db);
                break;
            case 3:
                do_promote(employee_db);
                break;
            case 4:
                employee_db.display_all();
                break;
            case 5:
                employee_db.display_current();
                break;
            case 6:
                employee_db.display_former();
                break;
            default:
                std::cerr << "Unknown command.\n";
                break;
        }
    }
    return 0;
}

int display_menu()
{
    std::cout << std::endl;
    std::cout << "Employee Database" << std::endl;
    std::cout << "-----------------" << std::endl;
    std::cout << "1) Hire a new employee" << std::endl;
    std::cout << "2) Fire an employee" << std::endl;
    std::cout << "3) Promote an employee" << std::endl;
    std::cout << "4) List all employees" << std::endl;
    std::cout << "5) List all current employees" << std::endl;
    std::cout << "6) List all former employees" << std::endl;
    std::cout << "0) Quit" << std::endl;
    std::cout << "---> ";
    int selection;
    std::cin >> selection;
    return selection;
}

void do_hire(Database& db)
{
    std::string first_name, last_name;
    std::cout << "First name? ";
    std::cin >> first_name;
    std::cout << "Last name? ";
    std::cin >> last_name;
    auto& empolyee { db.add_employee(first_name, last_name) };
    std::cout << "Hired employee " << first_name << ", " << last_name << " with
employee number " << empolyee.get_employee_number() << std::endl;
}

```

```
void do_fire(Database& db)
{
    int employee_number;
    std::cout << "Employee number? ";
    std::cin >> employee_number;

    try {
        auto& e {db.get_employee(employee_number) };
        e.fire();
        std::cout << "Employee " << employee_number << " terminated." << std::endl;
    } catch (const std::logic_error& exception) {
        std::cerr << "Unable to terminate employee: " << exception.what() << std::endl;
    }
}

void do_promote(Database& db)
{
    int employee_number;
    std::cout << "Employee number? ";
    std::cin >> employee_number;

    int raise_amount;
    std::cout << "How much of a raise? ";
    std::cin >> raise_amount;

    try {
        auto& e {db.get_employee(employee_number) };
        e.promote(raise_amount);
        std::cout << "Employee " << employee_number << " get a raise of " << raise_amount << std::endl;
    } catch (const std::logic_error& exception) {
        std::cerr << "Unable to promote employee: " << exception.what() << std::endl;
    }
}
```

Employee类的定义参见如下的employee.hpp文件：

```
#ifndef employee_hpp
#define employee_hpp

#include <string>

namespace Records {

const int default_starting_salary { 30'000 }; //默认的初始工资
const int default_raise_and_demerit_amount { 1'000 }; //默认的工资增幅和降幅

class Employee {

```

```
public:
    Employee(const std::string& first_name, const std::string& last_name);
    void promote(int raise_amount = default_raise_and_demerit_amount); //增加工资，使用了默认实参，技术细节参考教材211页第6.5.1节
    void demote(int demerit_amount = default_raise_and_demerit_amount); //降低工资
    void hire(); //雇佣，成为在职员工
    void fire(); //解雇，成为离职员工
    void display() const; //显示员工基本信息

    void set_first_name(const std::string& first_name); //设置first name
    const std::string& get_first_name() const; //获取first name

    void set_last_name(const std::string& last_name); //设置last name
    const std::string& get_last_name() const; //获取last name

    void set_employee_number(int employee_number); //设置工号
    int get_employee_number() const; //获取工号

    void set_salary(int new_salary); //设置工资
    int get_salary() const; //获取工资

    bool is_hired() const; //获取在职状态

private:
    std::string m_first_name;
    std::string m_last_name;
    int m_employee_number { -1 };
    int m_salary { default_starting_salary };
    bool m_hired { false };
};

}
```

Database类的定义参见如下的database.hpp文件：

```
#ifndef database_hpp
#define database_hpp

#include <string>
#include <vector>
#include "employee.hpp"

namespace Records {

const int first_employee_number { 1'000 }; //初值员工工号

class Database {
public:
    Employee& add_employee(const std::string& first_name, const std::string& last_name); //在数据库中增加一个员工，并返回该员工的引用
```

```
    Employee& get_employee(int employee_number); //根据工号获取相应员工的引用，如果该工号对应的员工不存在，抛出一个异常logic_error，异常的相关技术细节参考教材172页第5.6节
    Employee& get_employee(const std::string& first_name, const std::string& last_name); //根据姓名获取相应员工的引用，如果该姓名对应的员工不存在，抛出一个异常logic_error，异常的相关技术细节参考教材172页第5.6节
    void display_all() const; //显示所有员工
    void display_current() const; //显示所有在职员工
    void display_former() const; ////显示所有离职员工

private:
    std::vector<Employee> m_employees; //存放员工数据
    int m_next_employee_number { first_employee_number }; //下一个员工的工号
};

}
```

请根据上述类的定义，完成相关方法的定义和实现。其中，employee.cpp文件提供Employee类的方法定义，database.cpp文件提供Database类的方法定义。然后编译运行userinterface.cpp文件来使用该员工管理系统。